Electronic Version
Stylesheet Version v1.1.1

Description

METHOD AND SYSTEM OF DATA TRANSFER FOR EFFICIENT MEMORY UTILIZATION

BACKGROUND OF INVENTION

[0001] Field of the Invention

[0002] The invention generally relates to the implementation of computer networks, and more particularly, to a method and apparatus for efficient utilization of memory and reduced memory waste in storing data blocks of network traffic.

[0003] Background Description

[0004] In networks that employ various network protocols such as, for example, the Internet Protocol (IP) stack family and Fibre Channel protocol, packets are sent and received of varying sizes to deliver a wide variety of contents and messaging information including, for example, data, video, voice, control information, etc. In order to process these packets at the receiving side, e.g., a network interface, various techniques are used to manage the incoming stream of varying sizes of packets.

[0005] Typical arrangements include employing some form of logic, either hardware centric or software centric, to receive and temporarily store the incoming packets in a memory area, e.g., a buffer pool or equivalent, until subsequent processing can access and retrieve the packets, as appropriate, for action on the packet contents. These arrangements may also employ distributed processing or centralized processing.

[0006] This interaction between a network interface logic and subsequent processing often employs various types of buffering arrangements to manage the sequencing of the packets in the buffer pools so that arrival sequences are maintained in time ordered, or otherwise, sequence ordered relationship to one another. Often packets will contain sequencing information so that re-arrangement can occur to place information in proper order as necessary.

[0007] Additionally, observation and analysis of network traffic in various applications shows that transmitted packets typically fall within ranges of relative sizes, (e.g., small packets or big packets, etc.). For example, for the Fibre Channel protocol, packets can range between a minimal packet size of 36 bytes to 2112 bytes. However, 50% of all packets exchanged are smaller than 100 bytes, whereas substantially the other half of packets are in the range of 1500 bytes and larger. Other protocols tend to exhibit similar statistical groupings of packet sizes and may have more groups or different ratios.

[0008] Typically. in network handlers interfaces, when packets arrive from the network at an input port, the hardware attached to the port usually communicates this information to a processor which performs routing actions and then passes the processed packet to an output port for delivery to its intended destination. The hardware attached to the input port is often a simple DMA circuit that receives the packet, writes it to memory, and then indicates to the controlling processor that a packet has been received.

[0009] The DMA controller is typically set up by using several registers which are written by the host processor. One of these registers contains the address of the memory area where the next packet received from the network is going to be stored. As the size of the packet is not known in advance, the assigned memory area is typically sized to fit the biggest packet allowed by the particular network protocol to avoid loss of data. As a result, memory is used inefficiently, as big chunks of memory area are assigned for storing small packets.

[0010] An alternative approach uses multiple linked smaller buffers for storing a received packet. If the received packet does not fit in a single small buffer, then several of those smaller buffers are linked together as a linked list to store a single packet. This results in better memory utilization, however, this introduces significant complexity in handling the packets.

[0011] Yet another approach uses dynamical allocation of memory for each packet. This requires lengthy searches to locate the sufficient available consequent memory block to fit the received packet, using one of the space allocating algorithms such as "first fit" or "best fit". Drawbacks of this approach are that the search and allocation of the memory is slow, requiring the memory to be accessed and analyzed before an allocation can be made, as well as need for storing allocation information.

SUMMARY OF INVENTION

[0012] In an aspect of the invention, a method is provided to reallocate buffer memory in a network device comprising the steps of allocating memory into at least a first portion and a second portion, creating a buffer pool in the first and second portion wherein each buffer in the first buffer pool may be equally sized and associated with a predetermined packet size and each buffer in the second buffer pool may be equally sized and associated with another predetermined packet size. the method further includes monitoring usage of buffers in each buffer pool to determine whether the number of occupied buffers in either buffer pool crosses a threshold and triggering a reallocation of memory when the number of occupied buffers in either buffer pool crosses a threshold.

[0013] In another aspect, a system for a network device that receives packets of variable length is provided comprising a memory having a first portion and a second portion, the first portion storing packets having a length less than a predetermined value and the second portion storing packets greater than a predetermined value. A memory system reallocates the memory of the first portion and the second portion when the memory of at least one of the first portion and the second portion crosses a threshold.

[0014] In another aspect of the invention, a computer program product is provided which comprises a computer usable medium having readable program code embodied in the medium that includes a first computer program code to allocate memory into at least a first portion and a second portion. A second computer program code creates a buffer pool in the first and second portion, each buffer in the first buffer pool may be equally sized and associated with a predetermined packet size and each buffer in the second buffer pool may be equally sized and associated with another predetermined packet size. Also included is a third computer program code to monitor unused buffers in each buffer pool to determine whether the unused buffers in either buffer pool falls below a threshold, and a fourth computer code to trigger a reallocation of memory when the unused buffers in either buffer pool falls below a threshold.

BRIEF DESCRIPTION OF DRAWINGS

[0015] The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of embodiments of the invention with reference to the drawings, in which:

[0016] Figure 1 is a block diagram showing an embodiment of an environment of the invention;

[0017] Figures 2A-2B are illustrations of allocated memory portions according to an embodiment of the invention;

[0018] Figure 3 is an illustration of a linked list during operation, according to an embodiment of the invention;

[0019] Figure 4 is an illustration of a linked list during reallocation, according to an embodiment of the invention;

[0020] Figures 5A-5B are illustrations of an embodiment of a table/list for managing buffers; and

[0021] Figures 6A - 6E are flow diagrams showing steps of using the invention.

DETAILED DESCRIPTION

[0022] This invention is directed to providing a flexible buffer management method and system for efficient utilization of memory and reducing memory waste in storing data blocks (e.g., packets) of network traffic. To implement the invention, in embodiments, after reception of a packet from the network interface, the packet is typically temporarily stored in an inbound buffer in the network interface from where it is then transferred to memory to await subsequent processing. This memory is allocated into two or more portions (i.e., buffer pools) according to packet sizes with buffers allocated in each portion suitable for storing packets according to the packet sizes associated with the portion. As packets arrive and are stored, buffers are used and released. If the number of used buffers in any portion crosses a threshold, a reallocation of memory from one portion to the other may occur. The invention further reduces memory waste due to the storing of all packets into the maximal sized memory chunks, and has benefits of simplicity associated with storing a packet in a single buffer instead of multiple linked buffers, thus offering better memory utilization and simple design. In addition, no memory search has to be made before an allocation is made.

[0023] Figure 1 is a block diagram showing an embodiment of an environment of the invention. A network device, generally shown as 100, typically includes a host processor and one or more applications 110. The network device 100 is connected to a network 120 for receiving and transmitting packets of data using various types of protocols via a network interface 130. The network interface 130 temporarily stores an incoming packet in its local memory until the incoming packet may be transferred via a direct memory access controller (DMA) (or equivalent) to a buffer pool such as small buffer pool 200 or large buffer pool 250. Any memory data structures (e.g., tables, linked lists, small buffer pool 200, large buffer pool 250, counters, timers, etc.) contained in the memory are typically under control of one or more memory manager components 160 as part of the host processor 110 complex, but may involve distributed architectures.

[0024] The memory buffer pools may be allocated in relation to the types of packet sizes expected at a receiving network device 100. This may include two or more types of buffer pool sizes with multiple buffers in each buffer pool. The buffers are typically sized according to traffic statistics obtained from communications over the network 120. This buffer sizing leads to optimization of memory utilization which is directly related to overhead costs such as memory costs, speed performance, packet processing reliability, and the like.

[0025] To transfer packets from the network interface to the memory buffer pools 200 or 250, a check is made to determine which size of buffer is needed to store the incoming packet and an appropriate sized buffer is acquired from the appropriate buffer pool. If a request for a buffer of a particular size causes the number of available free buffers in the pool to fall below a predetermined threshold, then a re-allocation of one or more buffers from one pool of a particular size to another buffer pool of a different size is triggered according to the invention.

[0026] By way of example, a DMA 140 may be used in the illustration of the dynamic nature of controlling buffer pools in memory, according to the invention. Once the DMA 140 is initialized by the host processor 110 and provided references to the small and large buffer pools 200 and 250, respectively, incoming packets are transferred by the DMA to the buffer pool for the appropriate sized packet. As the DMA transfers a packet into a small or large buffer, a memory manager 160 updates the DMA pointers to the next available free buffer area for the small or large buffer pool as appropriate. In other words, the memory manager keeps the DMA informed of where the next available free buffer area is available for either the next small or large packet.

[0027] Although the invention is described using two buffer size categories, i.e., small and large (or maximal), to illustrate embodiments of the invention, one of ordinary skill in the art would recognize that any number of buffer sizes and buffer pools could be utilized with appropriate modification to re-allocate buffers among multiple categories. Embodiments of the invention may be implemented with or without the use of linked lists, as necessary, for efficiency or preference considerations.

[0028] Figures 2A and 2B illustrate memory partitioning, according to an embodiment of the invention. Referring now to Figures 2A and 2B, two start addresses in memory may be provided by the memory manager 160 to the DMA (or equivalent), one pointing to a first portion of memory, which may be the small size buffer pool shown as reference numeral 200, and the other pointing to a second portion of memory which may be the large size buffer pool, shown as reference numeral 250. The relative size of each buffer pool is related to the expected ratio of amounts of packets associated with each buffer pool. For example, if half the number of packets are expected to be small size, then memory is allocated so that the number of small buffers (e.g., 210a and 210b) are substantially equal to the number of large buffers 270.

[0029] The sizes of the small and large buffers are predetermined and initialized during a startup and initialization process (or after a reset) of the network device 100 (or similar device) in which the invention provides memory management and reallocation. The buffer pool sizes are predetermined based upon statistical information of the packet sizes to be encountered during processing. However, over a period of time, historical data concerning changes in packet sizes may be collected. If a significant enough change occurs to the statistical base upon which buffer pools are currently sized, a re-sizing of buffer pools to re-allocate the buffers according to new statistical data may occur. This may translate into small size buffers becoming larger size buffers, or large size buffers may become smaller sized buffers. This may also mean that the overall total number of small buffers and large buffers may change accordingly as memory availability permits. Typically, the size of a large buffer is an integer multiple of a small buffer size.

[0030] In the examples of Figures 2A and 2B, ten small buffers 210a and 210b equal the size of one large buffer 270 as shown in Figure 2B. Figure 2A shows free small buffers 210a and occupied small buffers 210b. For ease of recognition, free buffers are shown with an F and occupied buffers with an O. Likewise, the large free buffers 270a are shown with an F and the occupied large buffers 270b are shown with an O. An occupied buffer contains a received packet of data waiting for further processing. A free buffer is available to receive a packet from the network interface 130.

[0031] In an embodiment, the buffers in a buffer pool are arranged in a linked list.3 shows an embodiment of a linked list 280 for managing buffers in a buffer pool such as the buffers of Figures 2A and 2B. Linked list operations is a data structure technique known to one of ordinary skill in the art and may involve various other configurations such as a doubly linked list or the like. The use of linked lists is but only one way to manage buffer pools. Other memory control structures may also be employed equally as well to track and manage buffers in buffer pools (e.g., bit-maps, lists, tables, etc) without departing from the scope of this invention.

[0032] The exemplary linked list shown in Figure 3 represents a linked list 280 corresponding to the small buffer pool 200 configuration as shown in Figure 2A. The memory manager 160, or equivalent, typically manages this linked-list. This link list 280 includes a free buffer pointer 287 to buffer No. 1 of the small buffer pool 200 which is currently the next free buffer that will be used when another small packet arrives. A free buffer counter 289 is also maintained to keep track of the current count of free small buffers as small buffers 210 are used and freed. In this example, eight free small buffers 210a are currently available. Careful traversal of the link list 280 shows the linkages throughout the linked list 280. Free small buffer No. 1 is linked to free small buffer No. 2, which, in turn, is linked to free small buffer No. 6, etc. The last free small buffer is No. 16 and contains a terminator, e.g., NULL. Other techniques exist for implementing linked lists, however, this linked list represents one embodiment.

[0033] A threshold count 286 is also shown in Figure 3. When the number of free (i.e., unused) small buffers falls below this threshold count (for this example, set to 4), a re-allocation of large buffers to small buffers occurs. Additionally, a threshold count exists for the large buffer pool and when falls below the threshold, causes re-allocation of small buffers to large buffers. The threshold, in embodiments may, alternatively, indicate how many buffers are occupied rather than free, but in either case, when buffers of a particular size category are running low on availability, a re-allocation occurs. In either case, a threshold is being crossed to indicate a type of buffer is running low on availability and may require re-allocation from another size buffer pool.

[0034] A corresponding link list and pointers also exists for the large buffer pool 250 but since its operation is parallel to the small buffer pool 200, it is not shown. The large buffer pool linked-list may also track free large buffers 270a and maintain a current count of free large buffers (alternatively, a count of occupied buffers may be maintained for either small or large buffers).

[0035] Dynamic Re-allocation of Buffers and Use of the Invention

[0036] The allocation of available memory area is performed dynamically. During the initialization of the system, the available memory area is partitioned into two sets of equal number of memory buffers, but where the size of buffer is small, for one set, and of the maximal packet, for the another set. However, this is dynamically changed during the running of the network application. If the number of available buffers in one set falls under a particular threshold, free memory area from the other set can be reassigned to the first set. To accomplish this, the buffer resizing is performed, i.e., big memory buffers are partitioned into a number of small buffers, and multiple small buffers are merged into a larger buffer. This dynamic reassignment of buffers makes system adaptable to sudden network traffic pattern change.

[0037] Whereas resizing of a big buffer into multiple small buffers is relatively straight forward, the merging of multiple small buffers into a big buffer is more complex, as all small buffers which are going to be merged have to be physically adjoined in the memory, and optionally, properly aligned. To solve this problem, an additional memory pointer is used, referred to as a reallocation boundary 230 (Figure 2). Upon initialization of resizing, this pointer is set to a memory address corresponding to the new maximum allocation range of the small buffer array. It should be noted that buffers could still be allocated beyond this boundary, but new buffers will only be allocated below this buffer. Since the usage of packets is dynamic, i.e., all buffers are reclaimed after a fixed time period (due to the packets either expiring or being transmitted). Thus, after a predetermined time period, all buffers beyond the allocation boundary should be guaranteed to be free and then may be merged into a large buffer and be included into the pool of large buffers.

[0038] Figure 4 is an embodiment of an intermediate stage of reallocation of small buffers to a large buffer and will be explained in conjunction with Figures 6A-6E. Figure 4 shows the reservation range of the ten small buffers 210, Nos. 1 - 10, denoted by the reallocation boundary 230. The reallocation boundary 230 is set for the new boundary of the small buffers, to reallocate small buffers 1 to 10 for a new big buffer. All new requests for small buffers should be allocated below the reallocation boundary 230. The free buffer pointer 287 is updated to reference the first free buffer below the reallocation boundary (buffer No. 11 in the figure), and free buffer counter 289 is updated. As new packets arrive, they will be stored in small buffers below the buffer labeled 10, i.e., in buffers no. 11, 14, 15 and 16. Note that some of the buffers 1 to 10 may be occupied when the reallocation process starts, but after a determined amount of time (i.e., the reallocation time periods) all of these small buffers are necessarily free.

[0039] Figure 5A is a table/list showing which small buffers 210 of Figure 2A are occupied. In embodiments, this table/list, generally shown as reference numeral 300, is used to record all occupied buffers in a buffer area. This is at least one alternative to linked-lists as a tracking mechanism of occupied/unoccupied buffers. A similar table/list (not shown) may exist to record occupied buffers in the large buffer pool 250. Using this table/list 300, in order to determine if all buffers from the allocated area (i.e., area to be reallocated) are free (i.e., unoccupied), this table is searched for an entry belonging to the allocated area. In the example of Figure 2A, this is the area demarcated by reference numeral 230, that is, the first ten small buffers 210. If such an entry is located in the table/list 300, there is at least one buffer still occupied, so that area can not be reallocated. The table/list 300 can be regularly polled until the area is free (or a timer employed), or it waits until the found entry is free before repeating the search of the table, or some other determinate is used for searching the table again. Once no entries belonging to the allocated area are found in the occupied buffer table, the allocated area can then be reallocated.

[0040] Figure 5B is a table/list showing which small buffers 210 of Figure 2A are free (i.e., unoccupied). In an embodiment, this free buffer table/list, generally shown as reference numeral 305, is used to record which small buffers 210 in the allocated area (i.e., area to be reallocated) are free (i.e., unoccupied). Again, this table/list 305 can be repeatedly scanned/searched at a predefined time interval, or for the first entry that has been determined to be missing (i.e., occupied), and then the search can be repeated for other missing entries. A similar table/list (not shown) may exist for the large buffer pool 250 and maintained in like fashion. This is at least one alternative to linked-lists. Table/list 300 and 305 may also represent a bitmap wherein each entry is a bit showing either free entries or occupied entries.

[0041] Figures 6A-6E shows steps of using the invention. Figures 6A-6E may equally represent a high-level block diagram of the invention implementing the steps thereof. The steps of Figures 6A-6E may be implemented on computer program code in combination with the appropriate hardware. This computer program code may be stored on storage media such as a diskette, hard disk, CD-ROM, DVD-ROM or tape, as well as a memory storage device or collection of memory storage devices such as read-only memory (ROM) or random access memory (RAM). Additionally, the computer program code can be transferred to a workstation over the Internet or some other type of network.

[0042] Figure 6A is flow diagram showing general steps of using the invention beginning at step 315 where memory is allocated into at least two portions, a first portion and a second portion based on predetermined packet sizes. The packet sizes typically are evaluated statistically into two or more groups as anticipated for a protocol environment. At step 320, buffer pools are created in each portion, each buffer in the pool equally sized for a predetermined packet size (or range of packet sizes) appropriate for the portion of memory. At step 325, the buffers are tracked to monitor free and/or used buffers in each portion independently as packets are stored and processed. A history of the actual packet sizes processed is maintained for possible eventual reallocation of memory portions if network traffic characteristics change.

[0043] Continuing at step 330, unused and/or used memory in each portion is monitored and a reallocation of memory from one portion to another portion is performed when the unused memory in one portion falls below a predetermined threshold. At step 335, reallocating may involve reserving one or more occupied buffers until they are free. This may also involve setting a timer to insure that the occupied buffer has sufficient time to be emptied according to an anticipated consumption rate of the incoming packets. The process then ends until a new allocation of memory is desired based upon a request for re-initialization or new statistical information on packet sizes. However, the dynamic operation of processing packets and reallocation of the portions of memory may continue as described in more detail as follows.

[0044] Referring to Figure 6B, beginning at step 350, the network device 100 receives a packet of data via the network interface 130 and is transferred to memory 150 by a DMA 140 (or equivalent). For each new DMA transfer, the memory manager checks the status registers for the DMA controller to obtain the information if a small or large packet buffer was used for the previous transfer, and then provides the new pointer to a new area (i.e., buffer) of corresponding size to the DMA controller, i.e., if the last transfer was to a small buffer, then a new small buffer address is provided.

[0045] Continuing at step 355, a check is made to determine whether the current received packet is a small or large packet. If a large packet, processing continues at LP (Figure 6C), otherwise if a small packet, a check is made at step 360 to see if the new small packet is about to cross a predetermined threshold count for the number of free versus occupied small buffers. At step 365, if the threshold count is being crossed a reallocation procedure is triggered.

[0046] Processing continues in either case to step 370 where a check is made to see if a small buffer is available (i.e., free). If a small buffer is not free, then at step 390, buffer overflow processing is initiated to perform a reset of an appropriate level, re-transmission, or other processing to deal with the overflow. If, however, a small buffer is free, then at step 375, the incoming packet is stored in the next available small buffer. At step 380, the linked list for the small buffer pool 200 is updated and the free buffer counter 289 is also updated. In another embodiment using an occupied buffer counter, the occupied buffer counter is updated. At step 385, packet history is updated for statistical tracking and possible subsequent memory apportionment if packet size characteristics change on the network. The process then suspends/ends until another packet arrives.

[0047] If a large packet (LP) was detected at step 355, then at step 400 (Figure 6C) a check is made to see if a threshold is being crossed for the number of large buffers now in use (i.e., an amount of unused memory associated with the large buffers has fallen below the predetermined threshold). If a threshold is being crossed then the number of large buffers needs to be increased, so at step 410, a reallocation is triggered to convert small buffers to a large buffer (as described below). If, however, a threshold has not been crossed at step 400, or the triggering at step 410 has occurred, then a check is made at step 420 to see if a large buffer is currently available. If none is available, then at step 460 a buffer overflow sequence is initiated to handle a buffer overflow. If a large buffer is available, then at step 430, the packet is stored in the next available large buffer, e.g., 270a. At step 440, the large buffer pool linked list is updated along with the large packet counts. At step 450, the packet history is updated for statistical reasons. The process then suspends/ends until another packet arrives.

[0048] Figure 6D is a flow diagram showing steps of an embodiment for reallocating a free large buffer 270a into multiple (e.g., 10) small buffers 210. This process was initiated in the flow diagram of Figure 6B, step 365, and is typically an asynchronous process. At step 500, a check is made to determine if a large buffer is available. If not, this reallocation process ends and no reallocation takes place. If, however, a large buffer is available, then at step 510 a check is made to see if the number of occupied large buffers is over a predetermined threshold. If over a predetermined threshold, then the process terminates and no reallocation takes place. If large buffers are available and the number of occupied large buffers are under a predetermined threshold, then at step 520, a large buffer is de-linked from the large buffer pool 250.

[0049] Continuing at step 530, the large buffer is broken into multiple smaller buffers. At step 540, the new small buffers are linked into the small buffer pool 200 thereby causing a reallocation of a large buffer into multiple small buffers. At step 550, counts of large and small buffers are updated (e.g., 289). This process may be repeated as necessary to adjust available small buffers or large buffers dynamically as packet traffic varies over time.

[0050] Figure 6E is a flow diagram showing reallocating free small buffers 210a into a large buffer triggered by step 410 of Figure 6C and typically occurring asynchronously. At step 600, a number of contiguous small buffers, equal to the amount of memory required to become one or more large buffers, are identified. The allocation boundary pointer 230 is set to this new value. At step 610, these small buffers are reserved as pending reallocation to a big buffer. Any reserved buffer, in embodiments, will not be used to receive any new packets. It is typically necessary to identify contiguous small packets sufficient to equal a large buffer size, as they must eventually be capable of receiving the data of a large packet.

[0051] At step 620, a check is made to see if all of the reserved small buffers 210 are currently free. If all are not free buffers (i.e., at least one is occupied), then at step 630 a pending reallocation timer is started, and the reallocation is suspended until the reallocation timer expires and processing resumes at step point 640. This reallocation timer may be used to suspend reallocation for a predetermined period of time sufficient to assure that all incoming packets in the small buffer pool have been processed by the host processor during a reservation period. In other words, the rate of consumption of the incoming packets is based on the network protocol speed and the processing rate associated with the host processor consumption of the incoming packets, therefore the reallocation timer is set to a value to assure all reserved small buffers are free (i.e., processed). Since the small packets were reserved, they were not used to receive any additional incoming packets, and since the reserved buffers were not used to receive any new incoming packets, they are guaranteed to be free.

[0052] The small buffer to large buffer reallocation process continues at step 650 (from either step 620 or step 640) where all the reserved free contiguous small buffers are removed and de-linked from the small buffer pool 200. At step 660, the reserved small buffers 210 are consolidated and linked into the large buffer pool 250 as one new large buffer 270. At step 670, large and small buffer counts are adjusted and statistical history is updated. The process ends until another reallocation is triggered.

[0053] In another embodiment, a buffer table for recording occupied buffers in the allocated area, may be used. Once an area for reallocation is determined, all occupied buffers in this area are recorded in the buffer table. As each buffer is freed due to packet transmission or aborted because of expiration, a corresponding entry from the table is removed. Once the table is empty, the allocated area is free to be reallocated.

[0054] While the invention has been described in terms of embodiments, those skilled in the art will recognize that the invention can be practiced with modifications and in the spirit and scope of the appended claims.